Incremental Verification with Error Detection, Diagnosis, and Visualization

Kai-hui Chang
Avery Design Systems

David A. Papa, Igor L. Markov, and Valeria Bertacco
University of Michigan, Ann Arbor

Invers is a fast incremental-verification system for physical-synthesis optimization that includes capabilities for error detection, diagnosis, and visualization. Using a new metric called the similarity factor, Invers can help engineers identify potential errors earlier in development. Invers employs traditional verification only when necessary to ensure completeness of the verification flow. It also provides an error visualization interface to simplify error isolation and correction.

THE GROWTH IN complexity of digital designs poses increasing challenges to the functional verification of a circuit. Digital systems are commonly released with latent bugs, and the number of such bugs is growing for each new design. Further exacerbating the verification problem in modern designs is the increasingly dominant role that interconnects play in contributing to signal delay and power consumption. Performance optimization in modern designs requires tremendous effort in physical synthesis,[1] and it employs powerful optimizations such as retiming.[2] Because bugs still appear in many EDA tools today,[3] verifying the correctness of the performed optimizations is crucial. Traditional techniques address this verification problem by checking the equivalence between the original design and the optimized version. This approach, however, verifies only the equivalence of two versions of the design after completion of several, or possibly all, of the transformations and optimizations. Unfortunately, such an approach is not sustainable in the long term, because it makes the identification, isolation, and correction of errors introduced by transformations extremely difficult and time-consuming. On the other hand, performing traditional equivalence checking after each circuit transformation is too demanding. Because functional correctness is the most important aspect in high-quality design, considerable effort is currently devoted to verification and debug, expending resources that could otherwise have been dedicated to improving performance. As a result, verification has become the bottleneck that limits which novel and improved features a design can include.[4]

Thus, addressing this verification bottleneck is critical to improving design quality. Recent advancements in this area often focus on improving the performance of the verification tool itself. For example, researchers have proposed several techniques that use simulation to accelerate equivalence checking based on satisfiability (SAT) or binary decision diagrams (BDDs).[5] We advocate not only investing in the performance of verification algorithms and tools but also revising the design methodology to ease the burden on verification and debug.

We have developed an incremental-verification system (Invers) that enhances the accuracy of error detection. Invers uses a new metric called the similarity factor to quickly pinpoint potential bug locations, exposing design errors earlier during the optimization flow and facilitating debugging. The high performance of our equivalence verification solution allows quick evaluation of the correctness of each design transformation. When an error is detected, Invers provides a counterexample so that the designer can analyze the error directly. Our technique also suggests the most probable location and source of the error, typically pinpointing the specific transformation responsible for it. To further improve designer productivity, Invers provides an intuitive GUI. Our techniques can greatly improve design quality. The resources and effort saved in verifying the correctness of physical optimizations can be

0740-7475/09/$25.00 © 2009 IEEE Copublished by the IEEE CS and the IEEE CASS

redirected to improve other aspects of the design, such as reliability and performance. In addition, more aggressive changes to the circuit can be applied, such as retiming optimizations and design-for-verification (DFV) techniques.

We built our implementation using the OpenAccess database (http://www.si2.org) and the OpenAccess Gear (OAGear) programmer's toolkit, so that we could seamlessly integrate design, verification, and debug activities into the same framework. This framework is highly flexible and can easily be enhanced.

Background

Invers addresses the functional verification of incremental netlist transformations. Here, we describe two common optimization techniques: physical synthesis and retiming. We also explain the challenges to verification imposed by these techniques.

Physical-synthesis flows

Postplacement optimizations have been studied and used extensively to improve circuit parameters such as power and timing, and these techniques are often referred to as *physical synthesis*. In addition, it's sometimes necessary to manually change the layout to fix bugs or optimize specific objectives; this process is called an *engineering change order* (ECO). Physical synthesis commonly involves the following steps:

- perform accurate analysis of the optimization objective,
- select gates to form a region for optimization,
- resynthesize the region to optimize the objective, and
- remove any overlaps between standard cells, introduced by resynthesis.

Because subtle, unexpected bugs still appear in physical-synthesis tools today,[3] verification is necessary to ensure that the circuit functions correctly. However, verification is typically time-consuming, so it's often postponed until hundreds of optimizations have been completed. Consequently, if an error is detected, pinpointing the specific circuit modification that introduced the bug is difficult. In addition, debugging a circuit at this design stage is often difficult because engineers are unfamiliar with the automatically generated netlist. Invers addresses these problems by providing a fast incremental-verification technique and an integrated error visualization tool.

Retiming

Retiming is a sequential logic optimization technique that relocates registers in a circuit while leaving the combinational cells unchanged.[2] Engineers often use retiming to minimize the number of registers in a design or to reduce a circuit's delay. Although retiming is a powerful technique, ensuring its correctness is a complex verification problem. Sequential equivalence checking is far more difficult than combinational equivalence checking.[6] Consequently, if the analysis terminates, the runtime of sequential verification is typically far longer than that of combinational verification. In this article, we propose new techniques that extend our previous work to address the sequential-verification problem for retiming.[7]

Incremental verification

Our incremental-verification package consists of a logic simulator, a SAT-based formal equivalence checker, our innovative similarity metric between a circuit and its revision, and new visualization tools to aid users of our proposed incremental-verification methodology.

New metric: Similarity factor

We define an estimate of the similarity between two netlists, ckt_1 and ckt_2, which uses fast simulation. This estimate is the *similarity factor*. This metric is based on simulation signatures of individual signals—that is, the k-bit sequences holding signal values computed by simulation on each of k input patterns (for example, $k = 1{,}024$). Let N be the total number of signals (wires) in both circuits. Out of these N signals, we distinguish M matching signals. A signal is considered matching if and only if both circuits include signals with an identical signature. The similarity factor F_S between ckt_1 and ckt_2 is then M/N. We also define the difference factor F_D as $1-F_S$.

As an example, consider the two netlists in Figure 1, where the signatures appear above the wires. There are 10 signals in the netlists, and seven of them match. As a result, the similarity factor is $7/10 = 70\%$, and the difference factor is $1-7/10 = 30\%$.

Intuitively, the similarity factor of two identical circuits should be 100%. If a circuit is changed slightly but is still mostly equivalent to the original

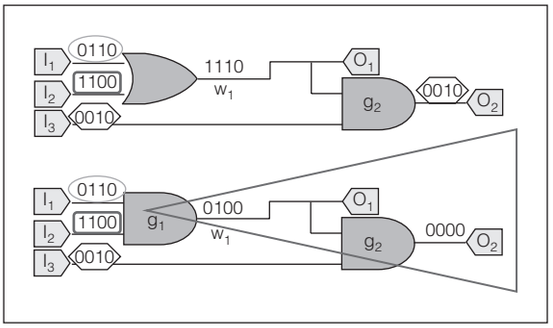

Figure 1. Similarity factor example. (Notice that the signatures in the corrupted signal's fan-out cone are different.)

version, then its similarity factor should drop only slightly. For example, Figure 2a shows a netlist in which a region of gates is resynthesized correctly. Because only the signatures in that region are affected, the similarity factor is still high. However, if the change greatly affects the circuit's function, the similarity factor can drop significantly, depending on the number of signals affected by the change. As Figure 2b shows, when resynthesis introduces a bug, the signatures in the resynthesized region's output cone are also different, causing a larger drop in the similarity factor. However,

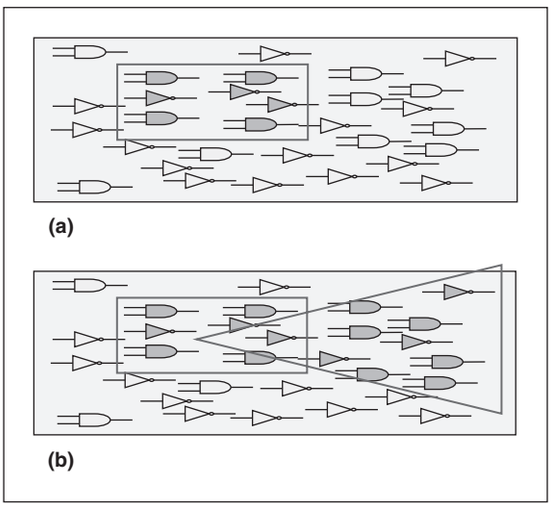

Figure 2. Resynthesis examples: the gates in the rectangle are resynthesized correctly, and only their signatures might differ from the original netlist (a); an error is introduced during resynthesis, leading to potential signature changes in the resynthesized region's fan-out cone, significantly increasing the difference factor (b).

two equivalent circuits can be dissimilar—for example, a carry-look-ahead adder and a Kogge-Stone adder. Therefore, the similarity factor should be used in incremental verification and not as a replacement for traditional verification techniques.

One issue that can affect the accuracy of the similarity factor is that two different signals can have identical signatures by coincidence. One way to counter this is to use more or higher-quality simulation vectors. For example, input vectors generated by ATPG tools typically have higher quality because such vectors have better signal-distinguishing capabilities than those generated by random simulation. An orthogonal but more effective technique hashes signatures as well as the input supports of signals: two signatures match only when their input supports are also the same. Additionally, we could consider the distances from inputs to the signals in hops (logic depth). If two signals are found at very different distances from the inputs, they are considered different even when their signatures are identical. If one signature is wrong, however, its entire fan-out will usually be wrong, and the chance of the entire fan-out cone matching existing signatures by coincidence is low. This phenomenon makes the similarity factor more accurate, even without extensions for input support and signal depth, for circuits with deeper logic. These extensions are useful primarily for shallow circuits or signatures that are close to outputs, and also to precisely locate a bug at the tip of an incorrect fan-out cone.

Sequential verification for retiming

A signature represents a fraction of a signal's truth table, which in turn describes the information flow within a circuit. Although retiming can change the clock cycle at which a signature is generated, the generated signatures should still be identical. Figure 3 shows a retiming example from our earlier work.[7] Figure 3a is the original circuit, and Figure 3b is the retimed circuit. A comparison of signatures between the circuits shows that the signatures in Figure 3a also appear in Figure 3b, although the cycles in which they appear might differ. For example, the signatures of wire w (bold-faced numbers in the figure) in the retimed circuit appear one cycle earlier than those in the original circuit because the registers were moved later in the circuit. Otherwise, the signatures of Figure 3a and Figure 3b are identical. This phenomenon becomes more obvious when the

circuit is unrolled, as Figure 4 shows. Because the maximum absolute lag in this example is 1, retiming affects only the gates in the first and last cycles, leaving the rest of the circuit unmodified. As a result, most signatures generated by the unaffected gates should not change.

On the basis of this observation, we extend our similarity factor to sequential verification, and we describe the *sequential similarity factor* as follows. Assume two netlists, ckt_1 and ckt_2, where the total number of signals (wires) in both circuits is N. After simulating C cycles, $N \times C$ signatures will be generated. Among those, we count M matching signatures. The sequential similarity factor F_{SS} between ckt_1 and ckt_2 is then M/NC.

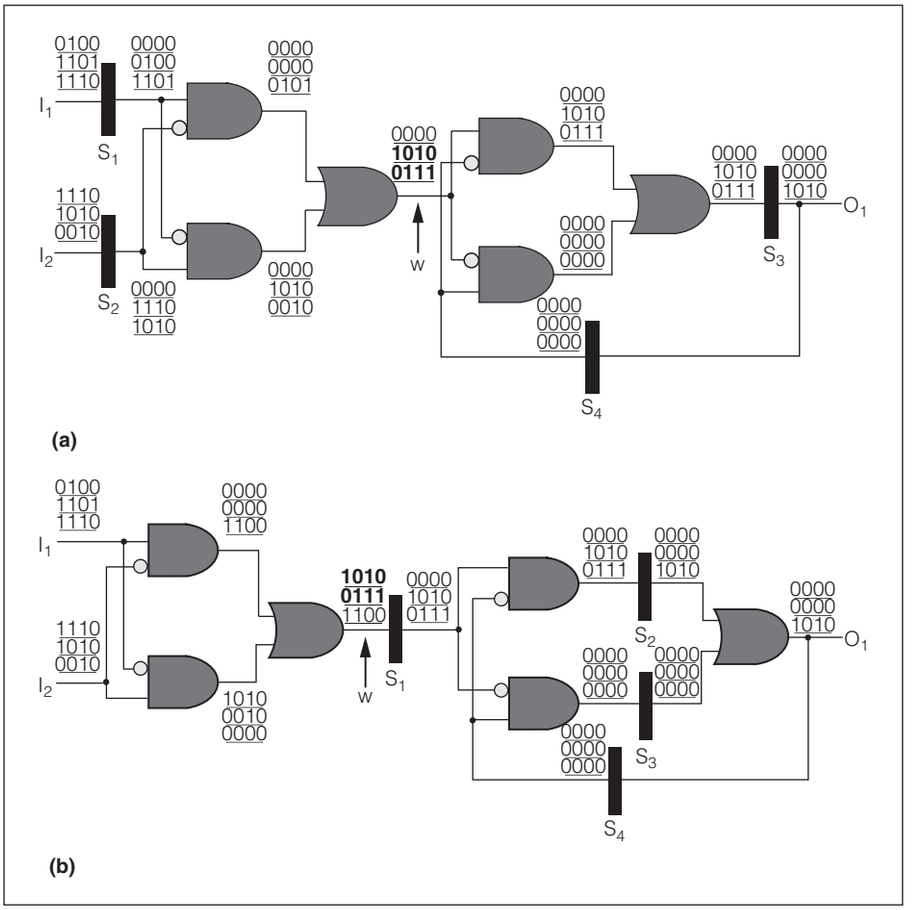

Figure 3. Retiming example: original circuit (a) and its retimed version (b). The tables above the wires show their signatures, where the *n*th row is for the *n*th cycle. Four traces generate the signatures, producing four bits per signature. Black rectangles represent registers, and the registers are initialized to 0. As wire *w* shows, retiming can change the cycle in which signatures appear, but not the signatures themselves. Corresponding signatures are shown in bold.

Error visualization

We compute the similarity factor by matching signals from two revisions of a design. Naturally, this process also identifies the nets that do not match the previous design version. Those nets are responsible for the change in circuit behavior. We use two techniques to highlight the differing nets, and in both cases we use a familiar layout format to present the results to the user. Our first technique uses a highlight color for the gates whose output signals have unmatched signatures. In the presence of an error, the layout highlights the entire output cone of the unmatched logic. The second technique highlights only the source of a problem—marking only those gates with matched input signatures and unmatched output signatures. Figure 5 shows an example of our visualization techniques. Notice that fixing these bugs could also unmask other bugs. For example, in Figure 5b we could detect only four of the five injected bugs. The two visualization techniques we have described let engineers quickly narrow down errors so that they can find the errors' original cause, which should simplify the correlation to the synthesis optimization that generated those errors.

Overall verification methodology

Traditional verification typically occurs after a batch of circuit modifications, because it is demanding and time-consuming. Therefore, once a bug is found, isolating the specific change that introduced the bug is difficult, because hundreds or even thousands of changes have been processed since the last check. The similarity factor addresses this problem by pointing out the changes that might have corrupted the circuit. A change that greatly affects the circuit's function will probably cause a steep drop in the similarity factor. By monitoring changes in the similarity factor after each circuit modification, engineers can isolate the circuit change that introduced a bug and trigger full equivalence checking immediately. On the basis of the techniques we've

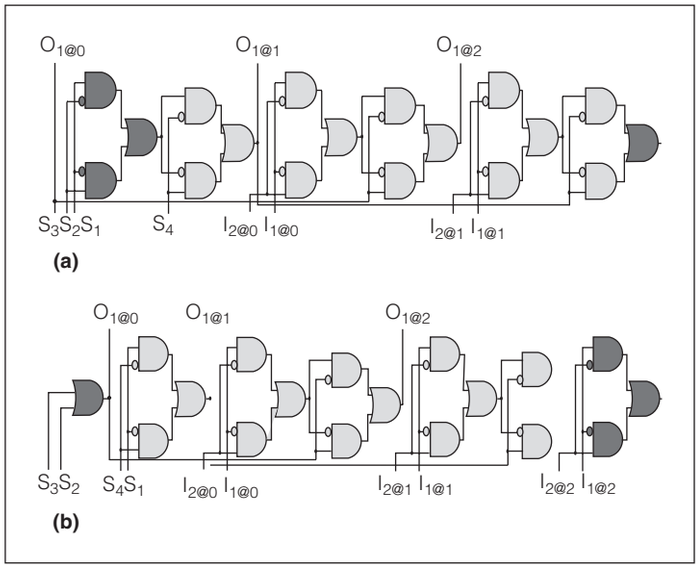

Figure 4. Circuits in Figure 3 unrolled three times. We use subscript @ to denote the cycle at which a signal appears. Retiming affects gates in the first and last cycles (dark gray), while the other gates are structurally identical (light gray). Therefore, only the signatures of the dark gray gates are different.

developed, we propose the following Invers verification methodology (see Figure 6).

First, after each circuit modification, calculate the similarity factor between the new and original circuit. Use the running average and standard deviation of the past 30 similarity factors to determine whether the most recent similarity factor is significantly lower. Empirical evidence suggests that if the latest similarity factor drops below the average by more than two standard deviations, the change likely had introduced a bug. This value, however, can vary among different benchmarks and should be empirically determined.

Second, when the similarity factor indicates a potential problem, use traditional verification to verify the correctness of the executed circuit modification.

Third, if verification fails, use the error visualization tools described in this article to debug the circuit by highlighting the gates producing differing signals.

Because Invers monitors the drops in (rather than the absolute values of) similarity factors, the netlist structures are less relevant. Therefore, Invers can be applied to various netlists, potentially with different error-flagging thresholds.

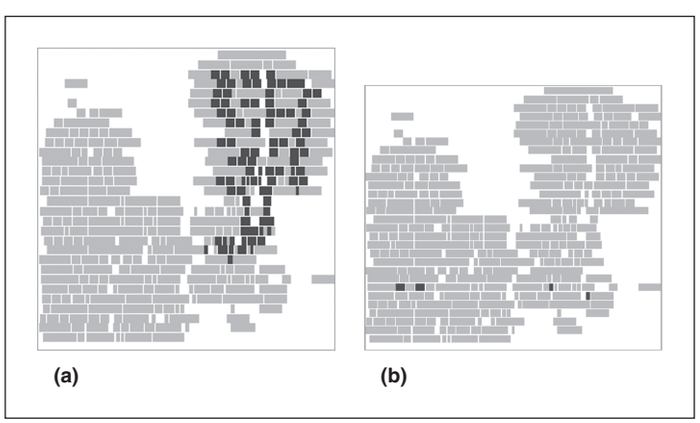

Figure 5. Our similarity layout viewer for the SASC benchmark design with bug-related data shown in bold: one bug injected, and highlighted gates drive unmatched signals (a); five unrelated bugs injected, and highlighted gates drive unmatched signals, but all their inputs are matched (four bugs are identified, and one is masked) (b).

Experimental results

We implemented Invers using OpenAccess 2.2 and OAGear 0.96 (http://www.si2.org). We selected our test cases from IWLS 05 benchmarks on the basis of designs from ISCAS 89 and OpenCores suites. Table 1 summarizes the characteristics of these benchmarks. We calculated the average logic depth by averaging the logic level of 30 randomly selected gates. The logic depth can serve as an indication of the circuit's complexity. We conducted all our experiments on an AMD Opteron 880 Linux workstation. The resynthesis package used in our experiments was ABC from the University of California, Berkeley.[8]

Verification for combinational optimizations

In our first experiment, we performed two types of circuit modifications to evaluate the similarity factor's effectiveness for combinational verification. In the first type, we randomly injected an error into the circuit according to Abadir's error model,[9] which includes errors that occur frequently in gate-level netlists. Injecting an error randomly mimicked a situation in which an optimization introduced a bug. In the second type of circuit modification, we extracted a subcircuit from the benchmark, containing 2 to 20 gates. We then resynthesized the subcircuit using ABC with the *resyn* command.[8] (This was similar to the physical synthesis or ECO flow we described earlier, in which gates in a small region of a circuit were modified.) Next, we used random simulation to generate 1,024 patterns and calculated the similarity

factor after each circuit modification for both types. We used 30 samples in this experiment. Table 2 summarizes the results.

As the results show, both types of circuit modifications decrease the similarity factor. However, the decrease is far more significant when we inject an error. The standardized differences between the means of most benchmarks (see d_1 in Table 2) are larger than 0.5, indicating that the differences are statistically significant. Because resynthesis tests represent the norm, and error-injection tests are anomalies, in our experiments we also calculated the standardized differences using only the standard deviations of resynthesis, SD_r (see d_2 in Table 2). As d_2 shows, for most benchmarks the mean similarity factor drops by more than two standard deviations when we inject an error. This result shows that the similarity factor is effective in predicting whether an optimization has introduced a bug. Nevertheless, in all benchmarks the maximum similarity factor for error-injection tests is larger than the minimum similarity factor for resynthesis tests, suggesting that the similarity factor cannot replace traditional verification and instead should be used as an auxiliary technique.

To evaluate the effectiveness of our incremental-verification methodology, we assumed one bug per 100 circuit modifications, and we calculated the accuracy of our methodology by measuring the fraction of cases in which the similarity factor correctly predicted equivalence. We also determined the runtime for calculating the similarity factor and the runtime for equivalence checking of each benchmark. Because most circuit modifications do not introduce bugs, we measured the runtime when equivalence was maintained. Table 3 summarizes the results.

Our methodology has high accuracy for most benchmarks. In addition, calculating the similarity factor is significantly faster than performing equivalence checking. For the largest benchmark (DES_PERF), for example, calculating the similarity factor takes less than 1 second, whereas performing equivalence

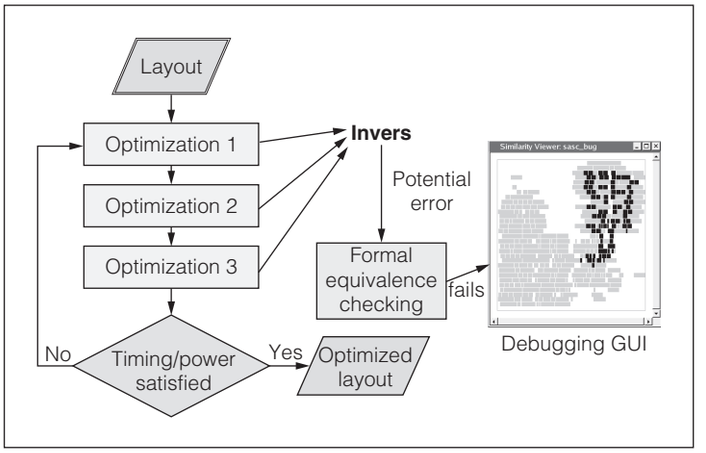

Figure 6. The Invers verification methodology monitors each layout optimization to identify potential errors, and invokes equivalence checking when necessary. Invers provides a GUI to support debug when verification fails.

Table 1. Benchmark characteristics.

Benchmark	Cell count	Average logic depth	Benchmark suite
S1196	483	6.8	ISCAS 89
USB_PHY	546	4.7	USB 1.1 PHY
SASC	549	3.7	Simple asynchronous serial controller
S1494	643	6.5	ISCAS 89
I2C	1,142	5.5	I2C master controller
DES_AREA	3,132	15.1	DES cipher (area optimized)
SPI	3,227	15.9	SPI IP
TV80	7,161	18.7	8-bit microprocessor
MEM_CTRL	11,440	10.1	Wishbone memory controller
PCI_BRIDGE32	16,816	9.4	PCI bridge
AES_CORE	20,795	11.0	AES cipher
WB_CONMAX	29,034	8.9	Wishbone Conmax IP core
DES_PERF	98,341	13.9	DES cipher (performance optimized)

* AES: Advanced Encryption Standard; DES: Data Encryption Standard; PHY: physical layer; SPI: serial parallel interface.

Table 2. Similarity factors (%) for resynthesis and error injection (one error injected).

	Resynthesis				Error injection					
Benchmark	**$Mean_r$**	**Min_r**	**Max_r**	**SD_r**	**Error**	**Min_e**	**Max_e**	**SD_e**	**d_1**	**d_2**
USB_PHY	99.849	99.019	100.000	0.231	98.897	91.897	99.822	1.734	0.969	4.128
SASC	99.765	99.119	100.000	0.234	97.995	90.291	99.912	2.941	1.115	7.567
I2C	99.840	99.486	100.000	0.172	99.695	98.583	100.000	0.339	0.567	0.843
SPI	99.906	99.604	100.000	0.097	99.692	96.430	99.985	0.726	0.518	2.191
TV80	99.956	99.791	100.000	0.050	99.432	94.978	100.000	1.077	0.930	10.425
MEM_CTRL	99.984	99.857	100.000	0.027	99.850	97.699	100.000	0.438	0.575	4.897
PCI_BRIDGE32	99.978	99.941	100.000	0.019	99.903	97.649	99.997	0.426	0.338	3.878
AES_CORE	99.990	99.950	100.000	0.015	99.657	98.086	99.988	0.470	1.372	21.797
WB_CONMAX	99.984	99.960	100.000	0.012	99.920	99.216	99.998	0.180	0.671	5.184
DES_PERF	99.997	99.993	100.000	0.002	99.942	99.734	100.000	0.072	1.481	23.969

* *min*: minimal value; *max*: maximum value; *SD*: standard deviation; d_1: standardized differences in the means, calculated using the average of both SD_e and SD_r: d_2: standardized differences in the means, calculated using only SD_r.

checking takes about 78 minutes. Because of the high accuracy of the similarity factor, our incremental-verification technique identifies more than 99% of errors, rendering equivalence checking unnecessary in those cases and providing more than 100× speedup.

Sequential verification for retiming

In our second experiment, we implemented the retiming algorithm described by Leiserson and Saxe,[2] and we used our verification methodology to check the correctness of our implementation. This methodology successfully identified several bugs in our implementation. Most of the bugs resulted from incorrect netlist modifications when repositioning the registers, and a few bugs were from erroneous initial-state calculation. Examples of the bugs included an incorrect fan-out connection when inserting a register to a wire that already had a register; a missing or extra register; a missing wire when a register drove a primary output; and an incorrect state calculation when two or more registers were connected in a row.

To quantitatively evaluate our verification methodology, we ran each benchmark using the correct implementation and the buggy version to calculate their respective sequential similarity factors. We ran the simulation for 10 cycles, and we ran 30 test sets per benchmark. Table 4 summarizes the results.

The sequential similarity factors for retimed circuits are 100% for most benchmarks. As explained earlier, retiming should affect only a few signatures. Therefore, the drop in similarity factor should be very small, making the sequential similarity factor especially accurate for verifying the correctness of retiming. This phenomenon is also evident from Table 5, where the accuracy of our verification methodology is higher than 99% for most benchmarks.

Table 5 also shows the runtime of a sequential equivalence checker based on bounded model checking. Our methodology is clearly more beneficial for sequential verification than for combinational verification, because sequential equivalence checking requires far more runtime than combinational does. Because the runtime to compute the sequential similarity factor remains small, our technique can still be applied after every retiming optimization, thus eliminating most unnecessary sequential-equivalence-checking calls.

Table 3. The accuracy of our incremental-verification methodology, assuming one bug per 100 circuit modifications.

			Runtime (s)	
Benchmark	**Cell count**	**Accuracy (%)**	**EC**	**SF**
USB_PHY	546	92.70	0.19	<0.01
SASC	549	89.47	0.29	<0.01
I2C	1,142	95.87	0.54	<0.01
SPI	3,227	96.20	6.90	<0.01
TV80	7,161	96.27	276.87	0.01
MEM_CTRL	11,440	99.20	56.85	0.03
PCI_BRIDGE32	16,816	99.17	518.87	0.04
AES_CORE	20,795	99.33	163.88	0.04
WB_CONMAX	29,034	92.57	951.01	0.06
DES_PERF	98,341	99.73	4,721.77	0.19

* EC: equivalence checking; SF: similarity factor.

Table 4. Sequential similarity factors (%) for retiming with and without errors.

	Retiming without errors				Retiming with errors			
Benchmark	$Mean_r$	Min_r	Max_r	SD_r	$Mean_e$	Min_e	Max_e	SD_e
S1196	100.0000	100.0000	100.0000	0.0000	98.3631	86.7901	100.0000	3.0271
USB_PHY	100.0000	100.0000	100.0000	0.0000	99.9852	99.6441	100.0000	0.0664
SASC	99.9399	99.7433	100.0000	0.0717	99.9470	99.3812	100.0000	0.1305
S1494	100.0000	100.0000	100.0000	0.0000	99.0518	94.8166	99.5414	1.5548
I2C	100.0000	100.0000	100.0000	0.0000	99.9545	99.6568	100.0000	0.1074
DES_AREA	100.0000	100.0000	100.0000	0.0000	95.9460	69.1441	100.0000	6.3899

* *min*: minimal value; *max*: maximum value; *SD*: standard deviation.

Table 5. Accuracy of our verification methodology, assuming one bug per 100 retiming optimizations.

				Runtime (s)	
Benchmark	**Cell count**	**Register count**	**Accuracy (%)**	**SEC**	**SSF**
S1196	483	18	99.87	5.12	0.42
USB_PHY	546	98	99.10	0.41	0.34
SASC	549	117	95.80	5.16	0.56
S1494	643	6	99.47	2.86	0.45
I2C	1,142	128	99.27	2491.01	1.43
DES_AREA	3,132	64	99.97	49,382.20	14.50

* SEC: sequential equivalence checking; SSF: sequential similarity factor.

OUR METHODOLOGY AND algorithms promise to decrease the number of latent bugs released in future digital designs and to facilitate more aggressive performance optimizations, thus improving the quality of electronic design in several categories. Our future work will focus on improving the accuracy of similarity factors and providing more user-friendly error visualization tools, as well as improving the speed of computing similarity factors with incremental computation. When a netlist change alters a simulation signature, we can mark it invalid without recomputing it. When enough changes have been made, we can recompute the similarity factor only for invalid signatures. This will reduce computations by avoiding the calculation of simulation signatures for unchanged signals as well as repeated calculation of frequently changed signatures. ■

References

1. R.C. Johnson, "Future of Chip Design Revealed at ISPD," *EE Times,* 17 Apr. 2008; http://www.eetimes.com/showArticle.jhtml?articleID=207400313.
2. C.E. Leiserson and J.B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica,* vol. 6, nos. 1-6, 1991, pp. 5-35.
3. "Conformal Finds DC/PhysOpt Was Missing 40 DFFs!" ESNUG (E-mail Synopsys Users Group) 464, item 4, 30 Mar. 2007.
4. I. Chayut, "Next-Generation Multimedia Designs: Verification Needs," 43rd Design Automation Conf. (DAC 06), ACM, 2006; http://videos.dac.com/43rd/23_2/23-2.html.
5. Q. Zhu et al., "SAT Sweeping with Local Observability Don't-Cares," *Proc. 43rd Design Automation Conf.* (DAC 06), ACM Press, 2006, pp. 229-234.
6. J.-H.R. Jiang and R.K. Brayton, "On the Verification of Sequential Equivalence," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 22, no. 6, 2003, pp. 686-697.
7. K.-H. Chang et al., "InVerS: An Incremental Verification System with Circuit Similarity Metrics and Error Visualization," *Proc. 8th Int'l Symp. Quality Electronic Design* (ISQED 07), IEEE CS Press, 2007, pp. 487-492.
8. Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification," release 51205; http://www.cad.eecs.berkeley.edu/~alanmi/abc.
9. M.S. Abadir, J. Ferguson, and T.E. Kirkland, "Logic Verification via Test Generation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 7, no. 1, 1988, pp. 138-148.

Kai-hui Chang is a senior member of the technical staff at Avery Design Systems, a supplier of functional-verification products based in Andover, Massachusetts. He performed the work described in this article while pursuing a PhD in the Department of Computer Science and Engineering at the University of Michigan, Ann Arbor. His research interests include verification, error diagnosis, error correction, and logic synthesis. He has a BS and an MS in electrical engineering from National Taiwan University, and a PhD in computer science and engineering from the University of Michigan, Ann Arbor. He is a member of the IEEE and the ACM.

David A. Papa is pursuing a PhD in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. He is also a research intern at IBM Austin Research Laboratory, where he works on the placement-driven synthesis (PDS) physical-synthesis tool. His research interests focus on physical design, especially VLSI CAD placement. He has a BSE and an MSE, both in computer science and engineering, from the University of Michigan, Ann Arbor. He is a student member of the IEEE.

Igor Markov is an associate professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. His research interests include physical design and physical synthesis for VLSI, quantum information and computation, verification of digital circuits and systems, and combinatorial optimization. He has a BA in mathematics from Kiev University; an MA in mathematics from the University of California, Los Angeles; and a PhD in computer science from the University of California, Los Angeles. He is a member of the IEEE Computer Society and the American Mathematical Society, and is a senior member of the IEEE and the ACM.

Valeria Bertacco is an assistant professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. Her research interests focus on formal and semiformal design verification, especially full-design validation and digital-system reliability. She has a Laurea in computer engineering from the University of Padova, Italy; and an MS and a PhD in electrical engineering from Stanford University. She is a member of the IEEE and the ACM.

■ Direct questions and comments about this article to Igor Markov or Valeria Bertacco, EECS Dept., University of Michigan, 2260 Hayward St., Ann Arbor, MI 48109; {imarkov, valeria}@umich.edu.

For further information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/csdl.

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.
MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.
COMPUTER SOCIETY WEB SITE: www.computer.org
OMBUDSMAN: To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to help@computer.org.
CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.
AVAILABLE INFORMATION: To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES
***Computer*:** The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.
Periodicals: The society publishes 14 magazines, 12 transactions, and one letters. Refer to membership application or request information as noted above.
Conference Proceedings & Books: Conference Publishing Services publishes more than 175 titles every year. CS Press publishes books in partnership with John Wiley & Sons.
Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.
Technical Committees: TCs provide professional interaction in over 45 technical areas and directly influence computer engineering conferences and publications.
Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.
Certifications: The society offers two software developer credentials.
For more information, visit www.computer.org/certification.

IEEE
Celebrating 125 Years
of Engineering the Future

revised 5 Mar. 2009

EXECUTIVE COMMITTEE

President: Susan K. (Kathy) Land, CSDP*
President-Elect: James D. Isaak*
Past President: Rangachar Kasturi*
Secretary: David A. Grier*
VP, Chapters Activities: Sattupathu V. Sankaran†
VP, Educational Activities: Alan Clements (2nd VP)*
VP, Professional Activities: James W. Moore†
VP, Publications: Sorel Reisman†
VP, Standards Activities: John Harauz†
VP, Technical & Conference Activities: John W. Walz (1st VP)*
Treasurer: Donald F. Shafer*
2008–2009 IEEE Division V Director: Deborah M. Cooper†
2009–2010 IEEE Division VIII Director: Stephen L. Diamond†
2009 IEEE Division V Director-Elect: Michael R. Williams†
***Computer* Editor in Chief:** Carl K. Chang†

** voting member of the Board of Governors † nonvoting member of the Board of Governors*

BOARD OF GOVERNORS

Term Expiring 2009: Van L. Eden; Robert Dupuis; Frank E. Ferrante; Roger U. Fujii; Ann Q. Gates, CSDP; Juan E. Gilbert; Don F. Shafer
Term Expiring 2010: André Ivanov; Phillip A. Laplante; Itaru Mimura; Jon G. Rokne; Christina M. Schober; Ann E.K. Sobel; Jeffrey M. Voas
Term Expiring 2011: Elisa Bertino, George V. Cybenko, Ann DeMarle, David S. Ebert, David A. Grier, Hironori Kasahara, Steven L. Tanimoto

EXECUTIVE STAFF

Executive Director: Angela R. Burgess
Director, Business & Product Development: Ann Vu
Director, Finance & Accounting: John Miller
Director, Governance, & Associate Executive Director: Anne Marie Kelly
Director, Information Technology & Services: Carl Scott
Director, Membership Development: Violet S. Doan
Director, Products & Services: Evan Butterfield
Director, Sales & Marketing: Dick Price

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036
Phone: +1 202 371 0101 • Fax: +1 202 728 9614
Email: hq.ofc@computer.org
Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314
Phone: +1 714 821 8380
Email: help@computer.org
Membership & Publication Orders:
Phone: +1 800 272 6657 • Fax: +1 714 821 4641
Email: help@computer.org
Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan
Phone: +81 3 3408 3118 • Fax: +81 3 3408 3553
Email: tokyo.ofc@computer.org

IEEE OFFICERS

President: John R. Vig
President-Elect: Pedro A. Ray
Past President: Lewis M. Terman
Secretary: Barry L. Shoop
Treasurer: Peter W. Staecker
VP, Educational Activities: Teofilo Ramos
VP, Publication Services & Products: Jon G. Rokne
VP, Membership & Geographic Activities: Joseph V. Lillie
President, Standards Association Board of Governors: W. Charlton Adams
VP, Technical Activities: Harold L. Flescher
IEEE Division V Director: Deborah M. Cooper
IEEE Division VIII Director: Stephen L. Diamond
President, IEEE-USA: Gordon W. Day

Next Board Meeting:
5 June 2009, Savannah, GA, USA